**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Искусственные нейронные сети»**

**Тема: «Распознавание рукописных символов»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Киреев К.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2021

**Цель работы**

Распознавание рукописных символов.

Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования.

**Задачи**

* Ознакомиться с представлением графических данных
* Ознакомиться с простейшим способом передачи графических данных нейронной сети
* Создать модель
* Настроить параметры обучения
* Написать функцию, позволяющая загружать изображение пользователи и классифицировать его

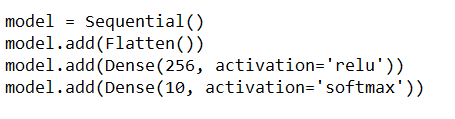
**Требования**

* Найти архитектуру сети, при которой точность классификации будет не менее 95%
* Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения
* Написать функцию, которая позволит загружать пользовательское изображение не из датасета

**Ход работы**

***Исследование архитектуры сети***

Была реализована архитектура нейронной сети, указанная в методических материалах.



При данной архитектуре нейронной сети точность результатов достигла 97.76%, что удовлетворяет требованиям лабораторной работы. Архитектура была оставлена без изменений.



***Исследование влияния оптимизаторов и их параметров***

Далее были рассмотрены различные оптимизаторы с разными параметрами и их влияние на точность предсказаний нейронной сети.

* ***SGD***

Стохастический оптимизатор градиентного спуска. Включает в себя поддержку импульса, затухания скорости обучения и импульса Нестерова.

* *learning\_rate*: скорость обучения
* *momentum*: параметр, ускоряющий SGD в соответствующем направлении и гасящий колебания
* *nesterov*: применять ли импульс Нестерова

|  |  |
| --- | --- |
| optimus = opti.SGD(learning\_rate=0.01, nesterov=False) | test\_loss: 0.32525  test\_acc: 91.07% |
| optimus = opti.SGD(learning\_rate=0.1, nesterov=False) | test\_loss: 0.13589  test\_acc: 96.06% |
| **optimus = opti.SGD(learning\_rate=0.1, nesterov=True)** | **test\_loss: 0.136**  **test\_acc: 96.12%** |

* ***RMSProp***

RMSProp оптимизатор. Рекомендуется оставить параметры этого оптимизатора на значениях по умолчанию (за исключением скорости обучения, которая может быть свободно настроена).

* *learning\_rate*: скорость обучения
* *rho*: коэффициент затухания скользящего среднего

|  |  |
| --- | --- |
| **optimus = opti.RMSprop(lr=0.001, rho=0.9)** | **test\_loss: 0.0714**  **test\_acc: 97.96%** |
| optimus = opti.RMSprop(lr=0.01, rho=0.9) | test\_loss: 0.21163  test\_acc: 96.18% |
| optimus = opti.RMSprop(lr=0.001, rho=0.6) | test\_loss: 0.08186  test\_acc: 97.70% |
| optimus = opti.RMSprop(lr=0.01, rho=0.6) | test\_loss: 0.20385  test\_acc: 97.40% |

* ***Adagrad***

Адаград — это оптимизатор, в котором скорость обучения зависит от конкретных параметров, которые адаптированы к тому, как часто параметр обновляется во время обучения. Чем больше обновлений получает параметр, тем меньше скорость обучения.

* *learning\_rate*: уровень начального обучения

|  |  |
| --- | --- |
| optimus = opti.Adagrad(learning\_rate=0.01) | test\_loss: 0.24117  test\_acc: 93.31% |
| **optimus = opti.Adagrad(learning\_rate=0.1)** | **test\_loss: 0.0921**  **test\_acc: 97.03%** |

* ***Adadelta***

Adadelta — более надежное расширение Adagrad, которое адаптирует скорость обучения на основе скользящего окна обновления градиентов вместо того, чтобы накапливать все градиенты прошлых лет. Таким образом, Adadelta продолжает обучение даже тогда, когда сделано много обновлений. По сравнению с Adagrad, в оригинальной версии Adadelta нет необходимости устанавливать начальную скорость обучения. В этой версии, как и в большинстве других оптимизаторов Keras, можно установить начальную скорость обучения и коэффициент распада.

* *learning\_rate*: начальная скорость обучения, по умолчанию 1. Рекомендуется оставить значение по умолчанию
* *rho*: коэффициент распада ададельты, соответствующий доле градиента, который необходимо сохранять на каждом шаге

|  |  |
| --- | --- |
| **optimus = opti.Adadelta(learning\_rate=1.0, rho=0.95)** | **test\_loss: 0.08035**  **test\_acc: 97.54%** |
| optimus = opti.Adadelta(learning\_rate=0.1, rho=0.95) | test\_loss: 0.21815  test\_acc: 93.76% |
| optimus = opti.Adadelta(learning\_rate=1.0, rho=0.6) | test\_loss: 0.09865  test\_acc: 97.19% |

* ***Adam***

RMSProp с моментами, получаемыми из предыдущих эпох.

* *learning\_rate*: скорость обучения
* *beta\_1*: обычно близко к 1
* *beta\_2*: обычно близко к 1

|  |  |
| --- | --- |
| **optimus = opti.Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999)** | **test\_loss: 0.07005**  **test\_acc: 97.79%** |
| optimus = opti.Adam(learning\_rate=0.01, beta\_1=0.9, beta\_2=0.999) | test\_loss: 0.15958  test\_acc: 96.63% |
| optimus = opti.Adam(learning\_rate=0.001, beta\_1=0.99, beta\_2=0.999) | test\_loss: 0.07533  test\_acc: 97.59% |
| optimus = opti.Adam(learning\_rate=0.001, beta\_1=0.999, beta\_2=0.999) | test\_loss: 0.09871  test\_acc: 97.07% |

* ***Adamax***

Это вариант Adam, основанный на норме бесконечности.

* *learning\_rate*: скорость обучения
* *beta\_1*: обычно близко к 1
* *beta\_2*: обычно близко к 1

|  |  |
| --- | --- |
| **optimus = opti.Adamax(learning\_rate=0.002, beta\_1=0.9, beta\_2=0.999)** | **test\_loss: 0.08871**  **test\_acc: 97.40%** |
| optimus = opti.Adamax(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999) | test\_loss: 0.11963  test\_acc: 96.58% |

* ***Nadam***

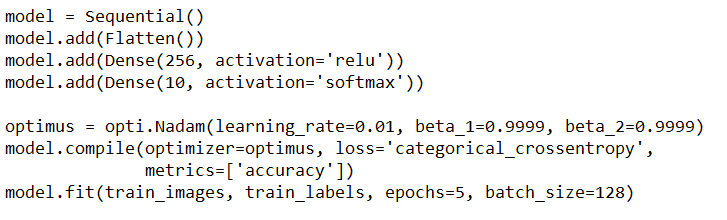
Nesterov Adam оптимизатор. Так же, как Adam, по сути, является RMSprop с импульсом, так и Nadam это RMSprop с импульсом Нестерова.

* *learning\_rate*: скорость обучения
* *beta\_1*: обычно близко к 1
* *beta\_2*: обычно близко к 1

|  |  |
| --- | --- |
| optimus = opti.Nadam(learning\_rate=0.002, beta\_1=0.9, beta\_2=0.999) | test\_loss: 0.07329  test\_acc: 97.72% |
| optimus = opti.Nadam(learning\_rate=0.01, beta\_1=0.999, beta\_2=0.999) | test\_loss: 0.13571  test\_acc: 97.75% |
| **optimus = opti.Nadam(learning\_rate=0.01, beta\_1=0.9999, beta\_2=0.9999)** | **test\_loss: 0.07958**  **test\_acc: 98.01%** |

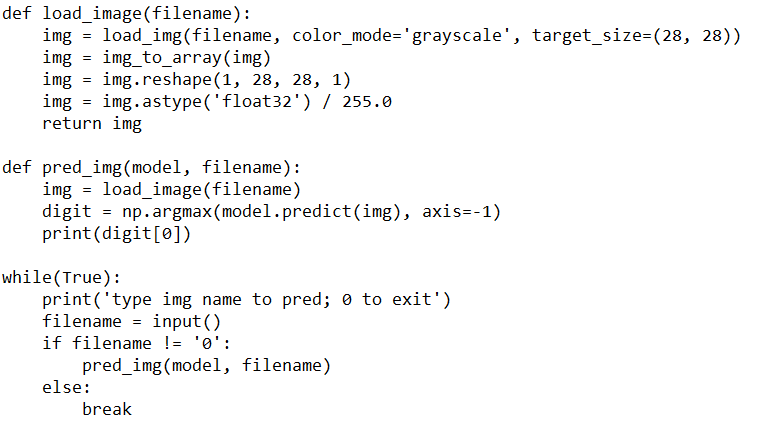
Лучшую точность показала нейронная сеть с оптимизатором *Nadam*, достигая точности 98.01%.

Конечные параметры нейронной сети:



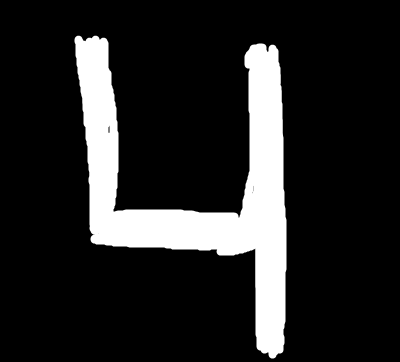
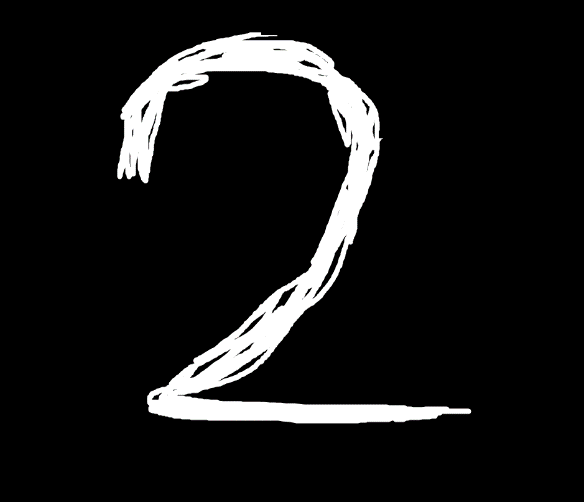
***Функция для загрузки пользовательских изображений***

Была написана функция для загрузки изображений пользователя.

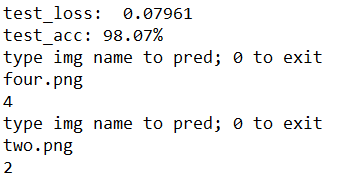


***Пример работы нейронной сети***

Изображения:



Вывод:



**Выводы**

В ходе выполнения лабораторной работы была создана нейронная сеть, которая классифицирует черно-белые изображения рукописных цифр по 10 категориям. Была найдена оптимальная конфигурация обучения сети, при которой сеть показывала максимальную точность. Была написана функция, позволяющая пользователю загрузить собственное изображение.